AMENDMENTS TO THE CLAIMS

Please amend the claims as indicated in the following listing of all claims:

1. (Original) A computer program product encoded in at least one computer readable medium, the computer program product comprising:

mutator code executable by a processor and including an instruction that coincides with a safe point in an execution path thereof, the instruction explicitly encoding parallelism amongst multiple component operations thereof,

wherein, based on a settable state of the processor, one of the operations of the instruction selectively triggers suspension of the mutator code at the safe point.

2. (Original) The computer program product of claim 1,

wherein the suspension triggering operation includes a conditional trap operation; and

wherein the settable state is encoded in storage accessed on execution by the conditional trap operation.

3. (Original) The computer program product of claim 1,

wherein the processor includes a very long instruction word-type (VLIW) processor and wherein the instruction is a VLIW instruction executable thereby; and

wherein the suspension triggering operation includes a conditional trap operation encoded in an otherwise unused position of the VLIW instruction.

4. (Original) The computer program product of claim 1,

wherein the mutator code is executable by the processor as plural execution threads each with respective safe points defined therein;

wherein the settable state is encoded in one or more storage locations accessible to the execution threads; and

wherein each of the execution threads includes an instance of the suspension triggering operation coinciding with respective of the safe points.

5. (Original) The computer program product of claim 1, wherein the mutator code is multi-threaded.

6. (Original) The computer program product of claim 1, further comprising:
suspension code executable by the processor and responsive to the triggering operation,
wherein upon execution, the suspension code advances the mutator code to the safe point.

7. (Original) The computer program product of claim 1,
wherein the suspension triggering operation is encoded at the safe point.

8. (Original) The computer program product of claim 1,
wherein the suspension triggering operation is encoded at a position in an execution sequence of the mutator code,
wherein the position precedes the safe point; and
wherein the mutator code is advanced to the safe point.

9. (Original) The computer program product of claim 1, further comprising:
suspension code executable by the processor and responsive to the triggering operation,
wherein upon execution, the suspension code supplies a collector with data that identifies zero or more storage locations containing pointers in use by the mutator code at the safe point.

10. (Original) The computer program product of claim 1, further comprising:
collector code executable by the processor to encode the settable state.

11. (Original) The computer program product of claim 10,
wherein the collector code includes first and second portions, the first portion ancillary to dynamic memory allocation, operation of the first portion causing the settable state to encode an exception triggering value in response to a garbage collection desired state of dynamic memory, and the second portion encoding garbage

collection operations to be performed after suspension of the mutator code at the safe point.

12. (Original) The computer program product of claim 1,
wherein the settable state is encoded in a register accessible to the processor on execution of the suspension triggering operation.

13. (Original) The computer program product of claim 1,
wherein the settable state is encoded in one or more bits of a processor status register.

14. (Original) The computer program product of claim 1,
wherein the at least one computer readable medium is selected from the set of a disk, tape or other magnetic, optical, electronic or semiconductor storage medium and a network, wired, wireless or other communications medium.

15. (Original) A method of suspending a mutator at a safe point, the method comprising:
executing mutator code including an instance of an instruction coinciding with the safe point, wherein the instruction instance references storage encodable with an exception triggering value to trigger an exception for suspending the mutator at the safe point;
executing the instruction instance without the exception triggering value encoded in the storage;
in response to a start garbage collection event, encoding the storage with the exception triggering value, and thereafter executing the instruction instance, thereby triggering the exception; and
in response to the exception, suspending the mutator at the safe point.

16. (Original) A method of suspending a mutator at a safe point, as recited in claim 15,
wherein the instruction instance coinciding with the safe point is encoded in an otherwise unused instruction position.

17. (Original) A method of suspending a mutator at a safe point, as recited in claim 16,

wherein the otherwise unused instruction position includes an otherwise unallocated instruction position within a very long instruction word-type (VLIW) instruction.

18. (Original) A method of suspending a mutator at a safe point, as recited in claim 16, wherein the otherwise unused instruction position includes a delay slot of a delayed control transfer instruction.

19. (Original) A method of suspending a mutator at a safe point, as recited in claim 15, wherein the instruction instance coinciding with the safe point is encoded in a VLIW instruction position.

20. (Original) A method of suspending a mutator at a safe point, as recited in claim 15, wherein the instruction instance coinciding with the safe point is encoded in a delay slot of a branch instruction.

21. (Original) A computer program product encoded in computer readable media, the computer program product comprising:

an instruction sequence including an operation that coincides with a safe point therein, wherein the operation is encoded in an otherwise unused position in the instruction sequence and is executable at least partially in parallel with one or more operations of the instruction sequence,

the operation referencing a state that selectively triggers suspension of the instruction sequence at the safe point.

22. (Original) The computer program product of claim 21,

wherein the instruction sequence includes a delayed control transfer instruction; and

wherein the otherwise unused position is a delay slot of the delayed control transfer instruction.

23. (Original) The computer program product of claim 21,

wherein the instruction sequence includes explicitly parallel instruction encodings; and

wherein the otherwise unused position is an operation position of one of the explicitly parallel instruction encodings.

24. (Original) The computer program product of claim 21,

wherein the instruction sequence includes very long instruction word-type (VLIW) instructions; and

wherein the otherwise unused position is an operation position of one of the VLIW instructions.

25. (Original) A computer-implemented method of providing mutator code with support for suspension of at least one execution thread thereof at a safe point, the method comprising:

encoding information accessible to a collector process to identify at least a portion of a root set of storage locations at a safe point in the execution sequence of instructions; and

encoding in an otherwise unused operation position of an instruction instance that coincides with the safe point, a conditional trap operation that references storage encodable with an exception triggering value to trigger an exception for suspending execution of the mutator code at the safe point.

26. (Original) A computer-implemented method as recited in claim 25,

wherein the identified portion of the root set includes zero or more registers and stack locations containing pointers in use by the mutator process at the safe point.

27. (Original) A computer-implemented method as recited in claim 26,

wherein a remaining portion of the root set includes storage maintained by an execution environment for the mutator code.

28. (Original) A computer-implemented method as recited in claim 25, further comprising:

compiling object code from source code, the conditional trap operation instance being encoded in the object code as a result of the compiling.

29. (Original) A computer-implemented method as recited in claim 25, further comprising:

supplying the execution sequence of instructions via at least one computer readable medium selected from the set of a disk, tape or other magnetic, optical, electronic or semiconductor storage medium and a network, wired, wireless or other communications medium.

30. (Original) A computer-implemented method as recited in claim 25, further comprising:

receiving a source code precursor of the mutator code into an execution environment therefor, the execution environment including a Just In Time (JIT) compiler,

wherein the conditional trap operation instance encoding and the information encoding are performed by the JIT compiler.

31. (Original) A computer-implemented method as recited in claim 30,

wherein the source code precursor receiving is via at least one computer readable medium selected from the set of a disk, tape or other magnetic, optical, electronic or semiconductor storage medium and a network, wired, wireless or other communications medium.

32. (Original) A computer program product comprising:

a compiler that generates an execution sequence of instructions including an explicit encoding of parallelism amongst component operations thereof, and which encodes in an otherwise unused component operation position an operation instance that references storage encodable with an exception triggering value to trigger an exception when the operation instance is executed as part of a mutator and to thereby suspend execution of the mutator at a safe point; and

a map generator that supplies a collector process with information identifying a root set of storage locations in use by the mutator for pointer storage at the safe point, the safe point coinciding with the operation instance.

33. (Original) The computer program product of claim 32, wherein the compiler includes one of:

a batch compiler; and

a just-in-time type (JIT) compiler.

34. (Original) The computer program product of claim 32, encoded by or transmitted in at least one computer readable medium selected from the set of a disk, tape or other magnetic, optical, semiconductor or electronic storage medium and a network, wireline, wireless or other communications medium.

35. (Original) A method of advancing plural threads to coordination points in respective execution paths thereof, the method comprising:

encoding an exception triggering value in storage referenced by respective instances of one or more operations encoded in respective otherwise unused operation positions in each of the plural threads;

for each of the plural threads, suspending execution thereof in response to execution of a respective operation instance, the respective operation instance coinciding with one of the coordination points therein.

36. (Original) A method, as recited in claim 35,

wherein the unused operation positions include delay slots of respective delayed control transfer instructions.

37. (Original) A method, as recited in claim 35,

wherein the unused operation positions include operation positions of respective explicitly parallel instruction encodings.

38. (Original) A method, as recited in claim 35,

wherein the unused operation positions include operation slots of respective very long instruction word-type (VLIW) instructions.

39. (Original) A method, as recited in claim 35, wherein the coordination points include synchronization points for thread state synchronization amongst the plural threads.

40. (Original) A method, as recited in claim 35, wherein the coordination points include safe points at which respective of the plural threads have a consistent state.

41. (Original) A method, as recited in claim 35, wherein the coordination points include safe points at which information descriptive of those temporary storage locations containing references to dynamically-allocated memory in the context of each function in a calling hierarchy of functions of a respective of the plural threads is ascertainable by a memory reclamation component for use in defining a root set of references to the dynamically-allocated memory.

42. (Original) A method of coordinating garbage collection with execution of a multi-threaded mutator, wherein the garbage collection is performed at safe points in an execution trajectory of the multi-threaded mutator, and wherein potentially inconsistent threads of the multi-threaded mutator are suspended at the safe points to facilitate the garbage collection, the method comprising:

upon a start garbage collection event, encoding an exception triggering value in storage referenced by exception triggering instructions in otherwise unused operation slots of instruction encodings that coincide with the safe points;

thereafter, upon execution of the exception triggering instructions, suspending the corresponding one of the threads; and

performing the garbage collection after each of the threads is suspended.

43. (Original) A method, as recited in claim 42, wherein the instruction encodings include one of:

component operations of a very long instruction word-type (VLIW) instruction; and

a pipelined first operation and corresponding delay slot operation.

44. (Original) A method, as recited in claim 42, wherein each of the threads is consistent when suspended at its corresponding safe point.

45. (Original) A method, as recited in claim 42, wherein the multi-threaded mutator is prepared such that dynamically allocated storage locations reachable by a particular of the threads are identifiable at the safe point thereof.

46. (Original) A method, as recited in claim 42, wherein the multi-threaded mutator is compiled and includes storage maps emitted by a compiler, the storage maps identifying dynamically allocated storage locations reachable by the multi-threaded mutator at the safe points.

47. (Original) An apparatus comprising:

a processor having a settable state accessible to plural execution threads thereon and having an instruction set that includes an exception triggering operation encodable in an operation position of an instruction encoding for parallel execution of component operations;

media accessible by the processor for encoding mutator code that includes an instance of the instruction encoding with an instance of the exception triggering operation encoded therein, the exception triggering operation instance referencing the settable state; and

a handler responsive to execution of the exception triggering operation instance when the settable state encodes the exception triggering value.

48. (***Currently Amended***) An apparatus as recited in claim 47, further comprising:

thread suspension means responsive to a start garbage collection event to encode the settable state with an exception triggering value~~; and~~ [.]

49. (Original) An apparatus as recited in claim 47,

wherein the exception triggering instruction is selected from the set of a trap on condition code instruction, a tagged arithmetic instruction, a precise interrupt generating instruction, an exception triggering instruction used in the mutator code solely for thread suspension, and an exception triggering used in the mutator code for purposes other than thread suspension but with support in the exception handler for distinguishing between uses.

50. (*New*) An apparatus as recited in claim 47,
wherein the exception triggering operation instance coincides with a safe point in an execution sequence of the mutator code.

51. (*New*) An apparatus as recited in claim 47,
wherein the exception triggering operation instance is encoded as component operation of a very long instruction word-type (VLIW) instruction.

52. (*New*) An apparatus as recited in claim 47,
wherein the instruction encoding for parallel execution includes a pipelined first operation and corresponding delay slot operation; and
wherein the exception triggering operation instance is encoded as a delay slot operation.